Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document No. 40

AN ILLIAC IV GAUSSIAN ELIMINATION PACKAGE FOR NON-CORE-CONTAINED MATRICES

by

Susan Ann Pace

September 1, 1972

CAC Document No. 40

AN ILLIAC IV GAUSSIAN ELIMINATION PACKAGE FOR NON-CORE-CONTAINED MATRICES

By

Susan Ann Pace

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

September 1, 1972

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the U.S. Army Research Office-Durham under Contract No. DAHC04-72-C-0001.

ABSTRACT

Gaussian Elimination is a method used to find the solution vector x of the equation $\underline{A}\,\underset{\sim}{x} = \underset{\sim}{b}$ where A is any general, non-singular, square matrix and $\underset{\sim}{b}$ is a vector. The process can be broken down into two independent phases:

(1) The triangular decomposition of the matrix $\underline{A}$, and

(2) The back substitution process to find the solution $\underset{\sim}{x}$.

This paper describes the Gaussian Elimination algorithm and the Illiac IV routines which perform the process.

AUTHOR'S NOTE

Until I/O specifications for ILLIAC IV are completed, the ASK programs described in this document will not be available for use. A revised document, including listings and output, will be produced at that time.

TABLE OF CONTENTS

1. GAUSSIAN ELIMINATION

The Gaussian Elimination Process (used to solve systems of simultaneous linear equations) involves reducing the (square) matrix of coefficients into an upper triangular matrix and performing back substitution to produce the solution. The reduction to an upper triangular matrix is done in N-1 steps (where N is the order of the matrix) as follows:

The system of equations is written as

$$\begin{array}{l} a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \ldots\ldots + a_{1n} x_n = b_1 \\ a_{21} x_1 + a_{22} x_2 + a_{23} x_3 + \ldots\ldots + a_{2n} x_n = b_2 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \qquad \vdots \\ a_{n1} x_1 + a_{n2} x_2 + a_{n3} x_3 + \ldots\ldots + a_{nn} x_n = b_n \end{array}$$

In forming the upper triangular matrix, the first derived system is found by multiplying the first row by a_{i1}/a_{11} (for $a_{11} \neq 0$) and subtracting the product from the ith row ($i > 1$). This produces:

$$\begin{array}{l} a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \ldots\ldots + a_{1n} x_n = b_1 \\ \qquad\quad a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \ldots\ldots + a_{2n}^{(1)} x_n = b_2^{(1)} \\ \qquad\quad \vdots \qquad\qquad\qquad\qquad\quad \vdots \qquad \vdots \\ \qquad\quad a_{n2}^{(1)} x_2 + a_{n3}^{(1)} x_3 + \ldots\ldots + a_{nn}^{(1)} x_n = b_n^{(1)} \end{array}$$

The new coefficients are given by

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \qquad i,j = 2, \ldots, n$$

$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} b_1 \qquad i = 2, \ldots, n$$

The second derived system is produced by multiplying the second row by a_{i2}/a_{22} (for $a_{22} \neq 0$) and subtracting the product from the ith row ($i > 2$). This results in:

$$\begin{array}{rcl}
a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \cdots\cdots + a_{1n} x_n &=& b_1 \\
a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \cdots\cdots + a_{2n}^{(1)} x_n &=& b_2^{(1)} \\
a_{33}^{(2)} x_3 + \cdots\cdots + a_{3n}^{(2)} x_n &=& b_3^{(2)} \\
\vdots \qquad\qquad \vdots && \vdots \\
a_{n3}^{(2)} x_3 + \cdots\cdots + a_{nn}^{(2)} x_n &=& b_n^{(2)}
\end{array}$$

The new coefficients are given by

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)} \qquad i, j = 3, \ldots, n$$

$$b_i^{(2)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)} \qquad i = 3, \ldots, n$$

Continue this process (through n-1 iterations) until the entire upper triangular matrix has been formed. At each step k, the new coefficients are found using the formulae:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \qquad i, j = k+1, \ldots, n$$

$$b_i^{(k)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)} \qquad i = k+1, \ldots, n$$

The final upper triangular matrix is:

$$\begin{array}{rcl}
a_{11} x_1 + a_{12} x_2 + a_{13} x_3 + \cdots\cdots + a_{1n} x_n &=& b_1 \\
a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \cdots\cdots + a_{2n}^{(1)} x_n &=& b_2^{(1)} \\
a_{33}^{(2)} x_3 + \cdots\cdots + a_{3n}^{(2)} x_n &=& b_3^{(2)} \\
\ddots \qquad \vdots && \vdots \\
a_{nn}^{(n-1)} x_n &=& b_n^{(n-1)}
\end{array}$$

The diagonal elements in this system must all be non-zero. If a diagonal element is found which is zero, that row must be interchanged with some subdiagonal row in the system which contains a non-zero element in that position. This interchange does not affect the solution. If no such row can be found, then the matrix is singular, and no solution is possible.

The back substitution phase of Gaussian Elimination computes the solution vector $\underset{\sim}{x}$. This is done using the formula:

$$x_i = \frac{1}{a_{ii}^{(i-1)}} \left[b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j \right] \qquad i = n, \ldots, 1$$

2. TRIANGULAR DECOMPOSITION

The Gaussian Elimination algorithm described in Chapter 1 is fine for matrices which are solved once and then forgotten. However, if the same matrix of coefficients is to be used with several right-hand sides (perhaps at different points in time), then it would be desirable to be able to find a solution without re-calculating the upper triangular form of the matrix. Since the same operations must be performed on the right-hand side as on the matrix of coefficients, one cannot merely save the upper diagonal form to use again.

The solution of this problem is to use the technique of triangular decomposition. Any nonsingular (square) matrix $\underline{A}$ can be decomposed into the product of a lower triangular matrix $\underline{L}$ and an upper triangular matrix $\underline{U}$ ($\underline{A} = \underline{L}\underline{U}$) where $\underline{U}$ is the matrix formed in the first phase of Gaussian Elimination and $\underline{L}$ contains the multipliers used in reducing $\underline{A}$ to $\underline{U}$. The lower triangular matrix $\underline{L}$ looks like:

$$\begin{matrix} 1 & & & & \\ \frac{-a_{21}}{a_{11}} & 1 & & & \\ \frac{-a_{31}}{a_{11}} & \frac{-a^{(1)}_{32}}{a^{(1)}_{22}} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \frac{-a_{n1}}{a_{11}} & \frac{-a^{(1)}_{n2}}{a^{(1)}_{22}} & \cdots\cdots & \frac{-a^{(n-2)}_{n,n-1}}{a^{(n-2)}_{n-1,n-1}} & 1 \end{matrix}$$

Furthermore, since the diagonal elements of $\underline{L}$ are all ones, they need not be stored. Thus, both $\underline{L}$ and $\underline{U}$ can be stored in a single n x n matrix, possibly overwriting $\underline{A}$.

Once $\underline{A}$ has been decomposed in this manner, a solution for any right-hand side $\underset{\sim}{b}$ can be found by solving the equations.

$\underline{L}\underset{\sim}{y} = \underset{\sim}{b}$ for $\underset{\sim}{y}$, and

$\underline{U}\underset{\sim}{x} = \underset{\sim}{y}$ for the solution $\underset{\sim}{x}$.

The solution $\underline{U}\underset{\sim}{x} = \underset{\sim}{y}$ is precisely the back substitution described above for Gaussian Elimination.

$\underline{L}$ and $\underline{U}$ are formed simultaneously from left to right one column at a time using the equations:

Column 1:

$$a_{11} = u_{11}$$

$$a_{i1} = \ell_{i1}\, u_{11} \qquad i = 2, \ldots, n$$

Column 2:

$$a_{12} = u_{12}$$

$$a_{22} = \ell_{21}\, u_{12} + u_{22}$$

$$a_{i2} = \ell_{i1}\, u_{12} + \ell_{i2}\, u_{22} \qquad i = 3, \ldots, n$$

Continue in this fashion to generate column r $(1 \le r \le n)$ using the general equation:

$$a_{1r} = u_{1r}$$

$$a_{2r} = \ell_{21}\, u_{1r} + u_{2r}$$

$$\vdots$$

$$a_{rr} = \ell_{r1}\, u_{1r} + \ell_{r2}\, u_{2r} + \cdots\cdots + \ell_{r,r-1}\, u_{r-1,r} + u_{rr}$$

$$a_{ir} = \ell_{i1}\, u_{1r} + \ell_{i2}\, u_{2r} + \cdots\cdots + \ell_{ir}\, u_{rr} \qquad i = r+1, \ldots, n$$

The above equations compute $\underline{L}$ and $\underline{U}$ simultaneously, column-by-column, and overwrite $\underline{A}$ as the decomposition is formed. No calculations are done on the right-hand side ($\underset{\sim}{b}$) at this point.

The complete separation of decomposition and the actual matrix solution allows the two phases to be coded as separate routines. This is advantageous since other matrix problems (notably inversion of a matrix) can proceed smoothly using $\underline{L}$ and $\underline{U}$ rather than $\underline{A}$.

There are two problems associated with triangular decomposition which need to be considered. The first is positioning for size. As mentioned in the discussion on Gaussian Elimination, all diagonal elements of $\underline{U}$ must be non-zero. This requires that a minimal amount of row interchanging be done. The second problem is scaling. Computers introduce some inherent error into any computation through round-off, truncation, etc. A matrix having a large range of magnitudes can quickly lose accuracy because of computer error. One way to reduce error is to use pivoting to assure the largest possible (in magnitude) divisor on the diagonals. Since complete pivoting requires extensive, messy coding, partial pivoting is generally used.

In partial pivoting, the diagonal and subdiagonal elements in any column are scanned, and the element having the largest magnitude is selected to be the diagonal (pivot) element. The row containing that element is interchanged with the row containing the diagonal position. Computation then proceeds normally. A record must be kept of all row interchanges to facilitate solving $\underline{L}\underset{\sim}{y} = \underset{\sim}{b}$. Also, because triangular decomposition is performed columnwise, two restrictions must be observed:

1) Do not interchange elements of the row to the left of the column being calculated.
2) Update a new column before searching for a pivot. (Step 1 of the sequential algorithm below).

To further reduce computer error, all calculations should be done using double precision arithmetic.

The algorithm for forming the triangular decomposition of a square matrix $\underline{A}$ on a sequential (conventional) computer follows below. The algorithm generates the r th column of $\underline{L}$ and $\underline{U}$, ($\underline{P}_j$ records the number of the row interchanged with row j to achieve partial pivoting in column j):

1) Place the double-length equivalent of column r into $\underset{\sim}{D}$ (a 1 x n double precision vector). For j = 1, ..., r-1 do the following:

   a) Take D_{p_j} , convert it to single precision to get u_{jr}, and overwrite a_{jr} with the result. Overwrite D_{p_j} with D_j.

   b) For i = j+1, ..., n subtract $\ell_{ij}\ u_{jr}$ from D_i using double precision arithmetic. Overwrite D_i with the result.

2) (Partial Pivoting). Select the largest D_i, i = r, ..., n, call it D_{p_r}, round it to single precision to get u_{rr}, and overwrite a_{rr} with u_{rr}. Store P_r and overwrite D_{p_r} by D_r.

3) For i = r+1, ..., n, divide D_i by u_{rr} to get ℓ_{ir} (which gives a single precision result) and overwrite a_{ir} with the result.

3. THE SOLUTION PHASE

The solution phase uses the decomposition of $\underline{A}$ and the right-hand side vectors $\underset{\sim}{b}_1, \ldots, \underset{\sim}{b}_m$ to find the solution vectors $\underset{\sim}{x}_1, \ldots, \underset{\sim}{x}_m$. The right-hand sides must first be updated to put them in the same form as the $\underline{U}$ matrix i.e., solve the equation $\underline{U}\underset{\sim}{y} = \underset{\sim}{b}$ for $\underset{\sim}{y}$. This is done as follows for each vector $\underset{\sim}{b}$:

Put the double length equivalent of b_i into D_i, $i=1, \ldots, n$.

For $i=1, \ldots, n$ do the following:

1) Round D_{p_i} to single precision to get y_i, overwrite D_{p_i} with D_i.

2) For $j = i+1, \ldots, n$ subtract $\ell_{ji}\, y_i$ from D_j and overwrite D_j using double precision arithmetic.

Now that the vectors $\underset{\sim}{b}_1, \ldots, \underset{\sim}{b}_m$ are in the same form as $\underline{U}$, the solutions can be found using the back substitution method. For each right-hand side, do the following (right-hand sides are still in double precision form in $\underline{D}$):

For $i=n, \ldots, 1$ perform the following calculations:

1) Divide D_i by u_{ii} to get x_i.

2) For $j = i-1, \ldots, 1$ subtract $u_{ji}\, x_i$ from D_j.

This forms the solution $\underset{\sim}{x}$ to the equation $\underline{\underline{A}}\underset{\sim}{x} = \underset{\sim}{b}$.

4. GAUSSIAN ELIMINATION PACKAGE ON ILLIAC IV

Up until now, we have been talking about sequential algorithms for solving systems of linear equations. The remainder of this report concerns two subroutines written for a parallel processor computer (Illiac IV) in the assembly language for that machine (ASK). Special considerations and problems related to the machine are discussed in detail.

5. SPECIAL TERMINOLOGY USED IN ASK DESCRIPTION

The following terms are used in the discussion of the ASK version of the Gaussian Elimination algorithm:

Matrix - the m x m input matrix, in the form described under "use of the Gaussian Elimination package".

Segment - an m x 64 piece of the matrix (i.e., 64 columns).

Tract - a 64 x m piece of the matrix (i.e., 64 rows).

Field - a 64 x 64 piece of the matrix (i.e. the intersection of a segment and a tract).

Order - a 1 x N matrix (where N is the order of the array) which contains the ordering information from the triangular decomposition phase. (i.e. order contains the record of all row interchanges made during triangular decomposition in order to achieve partial pivoting).

6. I/O ALGORITHMS FOR TRIANGULAR DECOMPOSITION PHASE

Because Illiac IV has a limited amount of memory, the matrices used in this routine are non-core-contained. This means that I/O methods are crucial. They must be the most efficient possible to reduce the amount of I/O wait time to a minimum. In the triangular decomposition phase, the unit of I/O is a field. This is the smallest amount of data read into core or written onto disk at any one time. Memory is divided into buffers or 64 rows each, allowing each buffer to exactly contain one field. Pointers are set up to each buffer, and a table is kept for fields containing information such as whether it is in memory, which buffer it is in, where on disk it is located, if there is an uncompleted read or write in progress, etc.

In triangular decomposition, the matrix is decomposed by segments. A segment is read in and updated by all the columns of the matrix. In order to reduce round-off errors, each field of the segment to be updated is processed using the fields to the left of it in that tract of L before the next field is started. The segment is then decomposed and written onto disk. In order to make updating tracts possible, row interchanges must involve the entire row (including elements of both U and L) instead of only the U elements of the row (which was done in the sequential version.) Thus, the third phase of processing a segment is to read in appropriate fields of each segment to the left and interchange the rows that were necessary for partial pivoting in the segment just processed.

In order to maximize the chances that information is in core when it is needed, the following read algorithm is initiated whenever a field buffer is freed. (DONEFLAG = non-zero indicates that all information needed to complete triangular decomposition has been read):

1) If DONEFLAG non-zero, go to step 12.

2) Test condition flag:

   If CONDITION all zeros, go to step 5.

   If CONDITION mixed, go to step 3.

   If CONDITION all ones, go to step 9.

3) (Read in segments to do row interchanging): Increment tract number by 1. (If greater than maximum tract in matrix, go to step 4.) Test that field of the segment being read. If it is already in core, repeat step 3. If not, read it, set up proper pointers, and go to step 12.

4) Increment segment number by 1. If the result equals the number of the segment just decomposed, then go to step 10. Otherwise, set the tract number equal to the number of the segment just decomposed and test that field of the current segment. If it is in core, go to step 3. Otherwise, read it, set up proper pointers, and go to step 12

5) (Read fields of L̲ in current tract for updating new segment prior to decomposition): If an unread field containing elements of L̲ cannot be found in this tract, go to step 6. Otherwise, read the left-most such field, set up proper pointers, and go to step 12.

6) Increment the tract counter. If the new counter is greater than the maximum tract in the matrix, go to step 7. Otherwise repeat step 5.

7) Set condition flag to mixed (i.e., next read will get segments for updating row interchanges). Set segment counter to 0. Set tract number equal to number of segments to be decomposed. If the field of the segment to be read (which is pointed to by the tract number) is in core, go to step 3. Otherwise read it, set up proper pointers, and go to step 12.

8) (Read the next segment to be decomposed that is not already in core): Increment tract counter. If it is greater than the maximum tract in the matrix, go to step 9. Otherwise, read that field of the segment, set up proper pointers, and go to step 12.

9) Set condition flag to all zeros (i.e., next read will begin reading tracts of L̲ to update segment to be decomposed). Set tract number to 0. If first field of first tract is in core, go to step 5. Else read it, set up proper pointers, and go to step 12.

10) If all segments yet to be decomposed have not been read, go to step 11. If all tracts needed to update the segment to be decomposed next have not been read, go to step 9. Otherwise, set DONEFLAG to non-zero and go to step 12.

11) Set condition flag to all ones (i.e. next read will begin reading a segment to be decomposed). Set tract number to 0. Increment counter for segments to be decomposed. Read first field of segment, set up proper pointers, and go to step 12.

12) End read routine.

The output algorithm is equally important. Fields are only written onto disk when necessary. Thus, the following rules cover output procedures:

1) While computing the decomposition of any segment, write each field onto disk as soon as it is completed.

2) While making row interchanges on segments previously decomposed, write any fields that were changed onto disk when all interchanges are complete on the segment.

The last important consideration for I/O is the algorithm used to free buffers. The rules governing this phase are as follows:

1) During the update phase (updating segment to be decomposed by previously calculated segments), fields of $\underline{L}$ used in the updating should be freed as soon as they are used.
2) During the row interchange phase (updating previously calculated segments by segment just decomposed) free unchanged fields of $\underline{L}$ as soon as all interchanges are made to the segment.
3) Free fields of the segment being decomposed only after all computations have been completed.
4) Free any field being ouput as soon as the write is completed.
5) Whenever a buffer is freed, initiate a new read into that buffer immediately.

7. TRIDEC - THE TRIANGULAR DECOMPOSITION PHASE

TRIDEC is an ASK subroutine which performs the triangular decomposition phase of Gaussian Elimination in a parallel manner on Illiac IV. In the following discussion, whenever operations on rows or columns of the matrix are referred to, the algorithm will perform the operation iteratively on up to 64 elements of the row or column simultaneously.

$\underline{L}$ and $\underline{U}$ are formed column-wise working from left to right through the matrix. Since the matrix is non-core-contained, only one segment of the matrix is assumed to be in core at any given time. The decomposition must therefore occur in three steps:

1) The update phase - For each segment $\underline{S}$ after the leftmost one, the following steps must be taken to bring the data up to date; i.e., the same operations must be performed on segments $\underline{S}$ as have been previously performed on segments 0, 1, ..., $\underline{S}$-1.

   a) Make all row interchanges in segment $\underline{S}$ that have previously been made in earlier segments in order to accomplish partial pivoting, i.e., for i = 0, 1, ..., $\underline{S} \otimes 64-1$ interchange row i with row ORDER(i) in the segment.

   b) Let I indicate which field of segment $\underline{S}$ is being updated. Let J represent the total number of tracts in the matrix (counting from 0). For I = 0, 1, ..., J do the following:

      Let K be the number of fields of $\underline{L}$ which are to the left of segment $\underline{S}$ (i.e., for tracts 0, 1, ..., S-1, we have k = 0,1,...,S-1. For tracts S, ..., j we have k = S-1). Let L represent which field of tract I is being updated by. Then for L = 0, 1, ..., K do the following:

Let Q indicate which column of field L is being used. Let R indicate which column of segment $\underline{S}$ is being updated. For R = 0, 1, ..., 63 do:

For Q = 0, 1, ..., 63 multiply element Q of field L of segment $\underline{S}$ by column Q of field L of tract I. Subtract the result from column R of field I of segment $\underline{S}$.

2) Decomposition Phase - The $\underline{U}$ elements in fields 0, 1, ..., S-1 are now computed. If only remains for us to compute the elements of $\underline{L}$ and the few elements of $\underline{U}$ located in field S. Let i represent the columns of $\underline{S}$. For i = 0, 1, ..., 63 do the following:

a) Find the maximum magnitude among elements $64 \otimes S+i$ through m of column i. Set order $(64 \otimes S+i)$ to the row number of that maximum element. Interchange rows $64 \otimes S+i$ and order $(64 \otimes S+i)$.

b) Divide the subdiagonal elements of column i by the new diagonal element

c) Let j represent the columns in $\underline{S}$ to the right of column i. For j = i+1, ..., 63, multiply the subdiagonal elements of column i by element $64 \otimes S+i$ of column j. Subtract the product from the corresponding elements of column j.

3) Row Interchange Phase - In order to update new segments tract-wise, all elements in rows must be interchanged. The newly-completed segment is being written onto disk. At this time, previous segments are updated. Let I indicate the segment being updated. Read in only fields S, S+1, ..., J of each segment. For I = 0, ..., S-1 do the following:

Let k indicate the row interchanges necessary. For $k= S \otimes 64$, ..., $(S+1) \otimes 64-1$ interchange row k and row ORDER(k) of segment I.

8. I/O ALGORITHMS FOR BACK SUBSTITUTION PHASE

As previously stated in the triangular decomposition discussion, having a non-core-contained matrix forces the I/O algorithms to assume a prominent position in the algorithm being coded. The back substitution phase poses less problems than the decomposition phase, but I/O still maintains a position of importance. Back substitution is done in two steps - manipulate a set of right-hand sides to correspond to the decomposed matrix $(\underline{L}\underset{\sim}{y} = \underset{\sim}{b})$, and perform back substitution to get a solution to the matrix $(\underline{U}\underset{\sim}{x} = \underset{\sim}{y})$. The read algorithm below assures that input wait time is minimal:

1) If DONEFLAG non-zero, then go to step 9. Otherwise go to step 2.
2) If last input was a field from $\underline{U}$, go to step 6. Otherwise go to step 3.
3) If no unread field of $\underline{L}$ exists in current tract, go to step 4. Else read first such field, set up proper pointers, and go to step 9.
4) If current tract is last tract in matrix, go to step 5. Otherwise increment tract counter by 1 and go to step 3.
5) Set flag to indicate reading fields of $\underline{U}$. Decrement tract counter by 1 and go to step 6.
6) If no unread field appears in current tract, go to step 7. Else read first such field, set up proper pointers, and go to step 9.
7) If current tract is tract 0, go to step 8. Else decrement tract counter by 1 and go to step 6.
8) Set DONEFLAG non-zero. Go to step 9.
9) End read algorithm.

Fields of $\underline{L}$ and $\underline{U}$ are never output. The right-hand sides of the matrix are read in segments of 64 columns each. The right-hand side is output when the back substitution is complete.

New fields are read whenever a buffer is freed. The rules governing the freeing of buffers are as follows:

1) All subdiagonal fields of L are freed as soon as they have been used.

2) The diagonal fields of L are freed as follows:

   a) If the matrix has 10 tracts or less, do not free the diagonals of L.

   b) If the matrix has between 11 and 17 tracts, free all but the last four diagonals.

   c) If the matrix has more than 17 tracts, free all but the last diagonal.

3) Free all fields of U immediately after use.

9. GAUSS - THE BACK SUBSTITUTION PHASE

GAUSS is an ASK subroutine which performs the back substitution phase of the Gaussian Elimination process in a parallel manner on Illiac IV. It assumes that the matrix to be solved has already been decomposed into $\underline{L}$ and $\underline{U}$ by TRIDEC.

The solution of the matrix is computed from $\underline{L}$ and $\underline{U}$ using the method of back substitution described above for Gaussian Elimination. Any number of solutions may be found --one for each right-hand side provided by the user. Up to 64 solutions can be found at one time. The results are obtained by working from bottom to top through the matrix. Elements of $\underline{U}$ are read in tract-wise from bottom to top. To minimize round-off error, only one field at a time of the right-hand side is involved with the calculation.

The solution is found in two steps. First, the right-hand side must undergo the same transformation as the original matrix did during the decomposition process described above. This is done by solving $\underline{L}\underset{\sim}{y} = \underset{\sim}{b}$ for $\underset{\sim}{y}$. Second, the back substitution mentioned above is carried out in order to find the solution. This is done by solving $\underline{U}\underset{\sim}{x} = \underset{\sim}{y}$ for $\underset{\sim}{x}$. These steps are described below:

1) Update right-hand sides: In order to perform back substitution, the right-hand sides must be permuted in the same way that the matrix was permuted in forming $\underline{U}$. The information necessary to do this permutation is contained in $\underline{L}$ and ORDER. The permuting of the right-hand sides $\underset{\sim}{b}$ is done in two phases. Let n represent the number of rows in $\underset{\sim}{b}$. Hence n-1 is the index of the last row of b. (Numbering begins at zero):

   a) For I = 0, ..., n-1 interchange row I of $\underset{\sim}{b}$ with row ORDER(I) of $\underset{\sim}{b}$. This aligns the rows in the same order as the rows in $\underline{L}$ and $\underline{U}$.

b) Solve $\underline{L}\underset{\sim}{y} = \underset{\sim}{b}$ for $\underset{\sim}{y}$. This is done by tracts of $\underline{L}$. Let m be the number of the last tract of $\underline{L}$: m = (n-1)/64. Let I be the tract counter. For I = 0, ..., m do the following:

Let J indicate fields within tract I. For J = 0, ..., I do the following:

Let K indicate the columns within field J of tract I of $\underline{L}$. For K = 0, ..., 63, do the following:

Let P indicate the columns of field I of the segment of right-hand sides for which a solution is currently being found. For I = 0, ..., Q (where Q = number of columns in the segment), do the following:

Multiply column K of field J of tract I of $\underline{L}$ by element P of row K of field J of the segment of right-hand sides and subtract the products from column P of field J of the segment of right-hand sides.

2) Back Substitution: The solution is found by performing back substitution using $\underline{U}$ and the array of permuted right-hand sides. It is computed tract-wise from bottom-to-top using the back substitution algorithm described below. There are two steps necessary in performing back substitution because the matrix is not core-contained.

For tracts I = m, ..., 0 of $\underline{U}$, do the following:

a) Update Phase: The current field of right-hand sides must be updated by that part of the solution already computed. Let S indicate fields of tract I. For J = I+1, ..., m do the following:

Let K indicate the columns within field J of tract I of $\underline{U}$. For K = 0, ..., 63, do the following:

Let P indicate the columns of field I of the segment of right-hand sides for which a solution is being computed. For P = 0, ..., Q (where Q is the number of columns in the segment), do the following:

> Multiply column K of field J of tract I of U̲ by element K of column P of field J of the segment of right-hand sides and subtract the products from column P of field I of the segment of right-hand sides.

b) Computation Phase: Let J indicate the columns of field I of tract I of U̲. For J = 63, ..., 0, do the following:

> Divide row J of field I of the segment of right-hand sides by element J of column J of field I of tract I of U̲. Let K indicate columns of field I of the segment of right-hand sides. For K = 0, ..., Q (where Q is the number of columns in the segment), do the following:
>
> > Multiply the superdiagonal elements of column J of field I of tract I of U̲ by element J of column K of field I of the segment of right-hand sides and subtract the products from the corresponding elements of column K of field I of the segment of right-hand sides.

10. USE OF THE GAUSSIAN ELIMINATION PACKAGE

Triangular Decomposition:

The triangular decomposition routine (named TRIDEC) is an ASK subroutine (called in the standard fashion) with these parameters:

MATRIX - an m x m (square) matrix ($m \geq n$). On entrance, it contains the matrix to be decomposed. The matrix is expected to be padded on the bottom and to the right with rows and columns of zeros so that m is a multiple of 64. (This is important in simplifying the coding so that execution time may be cut). It is expected that the matrix will be located on disk. It is further expected that the matrix was read onto disk in the following manner:

```
Store the matrix tract-wise by fields and store fields by
rows, i.e., let Q be the number of the last tract in the
matrix (Q = (n-1)/64). For tracts 0, ..., Q do the following:

    Let I indicate which field in the tract is being written.
    Let J indicate which row of field I is being written.
    For I = 0, ..., Q, do the following:

        For J = 0, ..., 63, write row J onto disk.
```

The actual parameter passed is the disk reference of the matrix. Upon completion of the algorithm, the matrix will have been overwritten and will contain L and U, i.e. the decomposition of the matrix.

N - The order of the matrix to be decomposed (before padding).

EPS - The desired cut-off value for determining the singularity of the matrix. If, after partial pivoting is done, the pivot element has a magnitude less than EPS, then the matrix is considered to be singular and computation ceases. It should be noted that since the absolute magnitude of the pivot is used, an EPS should be chosen which reflects the magnitude of the elements in the matrix. (e.g., an EPS of 10^{-5} may work fine with a matrix having elements of magnitude 10^{3}, but would be worthless in a matrix having elements of magnitude 10^{-6}).

<u>ORDER</u> - A 1 x m array which will keep track of all the row interchanges necessary to achieve partial pivoting. The actual parameter passed is a reference to the disk location where the data is stored.

<u>FAIL</u> - A label to which control is passed if the input matrix turns out to be singular.

Back Substitution:

The back substitution routine (named <u>GAUSS</u>) is an ASK routine (called in the standard fashion) with these parameters:

<u>MATRIX</u> - The m x m (square) matrix containing <u>L</u> and <u>U</u> which is the output of <u>TRIDEC</u>. The actual parameter passed is the disk reference of the matrix.

<u>N</u> - The same <u>N</u> used by <u>TRIDEC</u>.

<u>RIGHTSIDE</u> - An M x Q array containing P ($P \leq Q$) right-hand sides for which solutions are to be found. It is padded with columns and rows of zeros (as was <u>MATRIX</u>) so that Q is a multiple of 64. It is located on disk stored segment-wise (i.e., for segments 0, ..., R (R = (P-1)/64) store fields 0, ..., S (S = (N-1/64) onto disk by rows). The disk reference is the parameter passed to <u>GAUSS</u>. Upon completion of the subroutine, <u>RIGHTSIDE</u> will have been overwritten and will contain the solution vectors $\underset{\sim}{x}_0$, ..., $\underset{\sim}{x}_p$.

P - The number of right-hand sides for which solutions are desired ($1 \leq P \leq 2^{24} - 1$).

ORDER - The disk reference of the ORDER matrix generated by TRIDEC.

Notes About Usage:

TRIDEC and GAUSS are designed to handle large, general matrices. (Matrices of order $N \leq 1664$ are accepted). The following comments should be considered before using these routines:

1) If your matrix is of order $N \leq 64$, do not use these routines as they are not efficient for arrays of that size. A Gaussian Elimination routine is available which accepts only matrices of order $N \leq 64$.
2) If your matrix is of a special type (symmetric, sparse, etc.) there is probably a solution routine available for it which can take advantage of its special properties. GAUSS and TRIDEC cannot do this.
3) The algorithms are most efficient for matrices having an order $N < 600$. They will accept matrices up to $N = 1664$, but the closer one gets to that limit, the more time will be spent doing I/O. By the time 1664 rows are reached, most of the execution time will be spent in an I/O wait state. Thus a large matrix will need an enormous amount of time to execute.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Center for Advanced Computation University of Illinois at Urbana-Champaign Urbana, Illinois 61801	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

An ILLIAC IV Gaussian Elimination Package for Non-Core-Contained Matrices

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Research Document

5. AUTHOR(S) *(First name, middle initial, last name)*

Susan Ann Pace

6. REPORT DATE September 1, 1972	7a. TOTAL NO. OF PAGES 30	7b. NO. OF REFS
8a. CONTRACT OR GRANT NO. DAHC04 72-C-0001 b. PROJECT NO. ARPA Order No. 1899 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) CAC Document No. 40 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

Copies may be obtained from the address given in (1) above.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY U.S. Army Research Office-Durham Duke Station, Durham, North Carolina

13. ABSTRACT

Gaussian Elimination is a method used to find the solution vector $\underset{\sim}{x}$ of the equation $\underline{A}\ \underset{\sim}{x} = \underset{\sim}{b}$ where A is any general, non-singular, square matrix and $\underset{\sim}{b}$ is a vector. The process can be broken down into two independent phases:

(1) The triangular decomposition of the matrix $\underline{A}$, and

(2) The back substitution process to find the solution $\underset{\sim}{x}$.

This paper describes the Gaussian Elimination algorithm and the ILLIAC IV routines which perform the process.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Gaussian Elimination Triangular Decomposition Guassian Elimination Package on ILLIAC IV TRIDEC-The Triangular Decomposition Phase Back Substitution Mathematics of Computation Numerical Analysis Linear Algebra Matrix Algebra						